操作系统期中考前整理

二、运行环境和运行机制

1. 操作系统为什么需要硬件提供基本运行机制？

①处理器具有特权级别，能在不同的特权级别运行不同的指令集合；
②硬件机制可将操作系统和用户程序隔离。

2. 为什么操作系统要将指令集合划分为不同的特权级别？

有些指令错用会导致系统崩溃。为了防止不恰当的使用，操作系统将部分指令封装在内核中，确保其安全的情况下才允许执行。

3. 常见的特权指令和非特权指令

特权指令：启动 I/O，内存清零，修改程序状态字，设置时钟，允许/禁止中断，停机；
非特权指令：控制转移，算术运算，访管指令，取数指令。

4. CPU 状态之间的转换

用户态到内核态：通过中断/异常/陷入机制；
内核态到用户态：设置程序状态字。

5. 为什么要引入中断和异常？

引入中断是为了支持 CPU 和设备之间的并行操作；引入异常表示 CPU 执行指令本身遇到了问题。

6. 中断/异常机制的工作原理

处理中断/异常时，操作系统从用户态切换到内核态，硬件保存用户态的上下文环境，捕获中断源发出的中断/异常请求，将中断触发器内容按规定编码送入 PSW 相应位，硬件执行流程根据中断号，查中断向量表转移控制权给中断处理程序。软件识别中断/异常类型，保存相关寄存器信息，分析中断/异常具体原因，执行对应处理功能。执行完后，硬件恢复现场并返回被时间打断的程序。

7. 系统调用的作用

系统调用时操作系统提供给编程人员的唯一缺口，使 CPU 从用户态陷入内核态，从而允许用户在编程时调用操作系统的功能。

8. 系统调用机制的设计（为了实现系统调用，操作系统需要做什么）

①需要有中断/异常机制，支持系统调用服务的实现；
②用户需要选择一条特殊的指令（陷入指令，又称访管指令），引发异常，完成用户态到内核态的切换；
③内核需要为每个系统调用分配一个编号，成为是系统调用号；
④内核需要有系统调用表，存放系统调用服务例程的入口地址。

9. 系统调用的执行过程

①通过中断/异常机制，硬件保护现场，通过查中断向量表把控制权转给系统调用总入口程序；
②通过系统调用总入口程序，保存现场，将参数保存在内核堆栈里，通过查系统调用表将控制权转给相应系统调用处理例程或内核函数；
③内核执行系统调用例程；
④执行完毕后，内核恢复现场并返回用户程序。

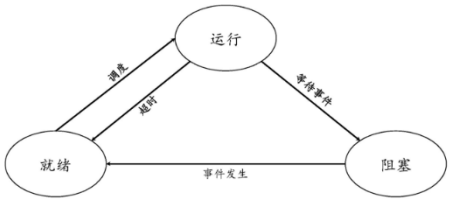
10. 系统调用和函数调用的区别、与 API 的关系

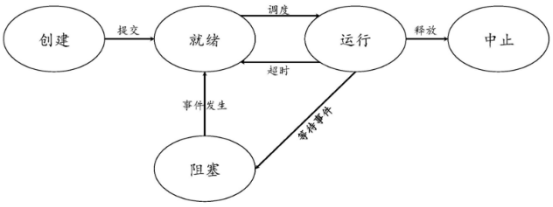
与函数调用的相同点在于都改变了指令的执行顺序，都必须返回原代码继续执行。区别：
①系统调用存在用户态和内核态的切换，调用地址不固定；
②系统调用中，内核使用不同的堆栈，存在堆栈的切换；
③系统调用不允许递归调用，函数调用允许递归调用；

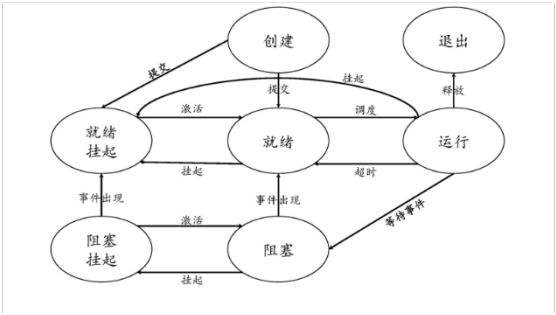
④系统调用通过 call 或 jmp 指令进入调用，函数调用通过 int 或 trap 指令进入调用；
与 API 的关系：API 是封装了的内核函数，API 执行过程可能经历了多次系统调用，多个 API 也可能都执行了同一个系统调用。

三、进程线程模型

1. 三、五、七状态进程模型







2. PCB 的作用

PCB（进程控制块）是操作系统中表示进程的数据结构，记录进程的各种属性，描述进程的动态变化过程，与进程一一对应，是系统感知进程存在的唯一方式。

3. PCB 的主要内容

包括进程描述信息（PID，进程名，用户标识符），进程控制信息（状态，优先级，代码执行入口地址…），所拥有的资源和使用情况（虚拟地址空间，打开文件列表），CPU 现场信息（寄存器值，页表指针）。

4. PCB 如何描述进程地址空间？

操作系统给每个进程都分配了一个地址空间。PCB 通过虚拟地址描述地址空间。

5. 进程是如何创建的？

①分配 PID，PCB；
②为进程分配地址空间；
③初始化 PCB；
④设置相应队列指针（将新进程添加进就绪队列链表中）；
⑤创建/扩充其他数据结构。

6. 进程是如何撤销的？

①结束子进程/线程；
②收回进程占有的资源（关闭打开文件，断开网络连接，回收分配的内存）；
③撤销 PCB。

7. Unix 的 fork()实现

①为子进程分配进程描述符、proc 结构和唯一的 PID；
②以一次一页的方式复制父进程地址空间（Linux 采用写时复制）；
③继承父进程的共享资源，包括打开文件和工作目录；

④将子进程状态设为就绪；
⑤为父进程返回子进程的 PID，子进程返回 0。

8. 为什么要引入线程？

①应用的需要：单进程无并行性；
②开销的考虑：创建线程花费时间少，线程切换花费时间少，线程之间相互切换无需调用内核；
③性能的考虑：多核，多处理器。

9. 进程和线程的区别

①进程是操作系统资源分配的基本单位，线程是处理器任务调度和执行的基本单位；
②进程都有自己独立的地址空间和资源，上下文切换开销大；线程共享地址空间和资源，上下文切换开销小；
③线程是进程的一部分，是轻量级的进程；
④一个线程崩溃后所有线程都会崩溃，而一个进程崩溃不会影响其他进程，多进程比多线程更加健壮；
⑤每个独立进程都有程序运行的入口，但线程不能独立执行，必须依附于进程中。

10. 线程的属性

①有状态和状态切换；
②不运行时，需要保存程序计数器等上下文；
③有自己的栈和栈指针；
④共享地址空间和资源；
⑤可以创建或撤销另一个进程。

11. 为什么线程要有自己的栈？

每个线程都独立执行，函数参数和局部变量都必须保存在栈中，所以每个线程要有独立的栈。

12. 线程的实现方式

线程的实现方式有用户级线程（在用户空间中实现）、核心级线程（在内核中实现），也有二者结合的方法（在用户空间创建，在核心态调度）。

13. 用户级线程和核心级线程的区别

①用户级线程在用户空间中建立，线程由用户管理，内核不知道线程的存在（只知道进程的存在）；核心级线程由内核管理，内核维护进程和线程的上下文；
②用户级线程切换不需要内核特权；核心级线程切换需要内核支持；
③典型例子：POSIX Pthreads 支持用户级线程，Windows 支持核心级线程。

四、进程线程调度

1. CPU 调度的时机

①进程执行完毕并退出；
②进程由于某种错误或异常而终止；
③新进程的创建；
④运行进程要等待 I/O 操作或其他资源时进入阻塞态；
⑤被唤醒的阻塞进程重新回到就绪态；
⑥进程用完所分配的时间片回到就绪队列。

2. 上下文切换的具体步骤（A 下 B 上）

①保存进程 A 的上下文环境；
②用新状态和其他信息更新进程 A 的 PCB；
③将进程 A 移动至合适的队列（就绪，等待…）；

④将进程 B 的状态设置为就绪态；
⑤从进程 B 的 PCB 中恢复上下文

3. 调度算法的衡量指标及主要考虑因素

衡量指标：公平性，吞吐量，周转时间，响应时间，CPU 利用率，等待时间。
主要考虑因素：PCB 中需要记录的信息，进程优先级及就绪队列的组织，抢占/非抢占，I/O 密集型/CPU 密集型，时间片。

4. 批处理系统中采用的调度算法

先来先服务（FCFS），最短作业优先（SJF），最短剩余时间优先（SRT），最高响应比优先（HRRN），其中响应比定义为

$$R=\frac{\text{作业周转时间}}{\text{作业运行时间}}=1+\frac{\text{作业等待时间}}{\text{作业运行时间}}.$$

5. 交互式系统中采用的调度算法

①轮转调度（RR）

时间片轮转调度：为每个进程分配一个时间片，在时间片结束时切换进程。优点：公平，响应时间快；缺点：由于进程切换，时间片轮转调度需要的开销大。

虚拟轮转调度：区分 CPU 密集型和 I/O 密集型。

②优先级调度算法（HPF）

选择优先级最高的进程优先执行，通常系统进程高于用户进程，前台进程高于后台进程，偏好于 I/O 型进程。

HPF 会出现优先级反转的现象，低优先级进程持有高优先级进程所拥有的资源，从而使高优先级进程等待低优先级进程。此时会导致系统错误，高优先级进程停滞不前导致系统性能降低。解决方案为设置优先级上限，优先级继承或中断禁止。

③多级队列调度算法（MQ）与多级反馈队列调度算法（MFQ）

设置多个优先级队列，第一级队列优先级最高。为不同队列分配长度不同的时间片（第一级最短），上一级队列为空时进入下一级队列调度，各级队列按照时间片轮转调度。若允许抢占，被抢占进程回到原一级队列队尾。

6. 各种调度算法的比较

调度算法	FCFS	RR	SJF	SRTN	HRRN	Feedback
选择函数	$\max\{w\}$	常数	$\min\{s\}$	$\min\{s-e\}$	$\max\left\{\frac{w}{s}\right\}$	队列
抢占	×	√	×	√	×	√
吞吐量	不强调	时间片小时低	高	高	高	不强调
响应时间	可能很高	短进程低	短进程低	低	低	不强调
开销	最小	最小	可能高	可能高	可能高	可能高
对进程影响	对短、I/O 进程不利	公平	对长进程不利	对长进程不利	平衡	可能对 I/O 进程有利
饥饿	×	×	√	√	×	√

五、进程同步机制

1. 进程的特征

①并发，共享，不确定性；
②程序执行结果的不可再现性；

③并发环境下进程间断执行；
④资源共享；
⑤独立性、制约性；
⑥程序和计算不再一一对应。

2. 进程互斥、临界资源、临界区

进程互斥：由于各要求使用共享进程，而这些资源需要排他使用，各进程之间这种竞争使用资源的关系称为是进程互斥。
临界资源：系统中某些一次只允许一个资源使用的资源称为是临界资源（互斥资源，共享变量）。
临界区（互斥区）：各个进程中对临界资源实施操作的程序片段。

3. 临界区的使用原则、前提

原则；
①有空让进：当没有进程在临界区时，任何有权使用共享资源的进程都有权进入临界区；
②无空等待：不允许两个以上进程同时进入临界区；
③有限等待：任何进入临界区的要求应该能在有限时间内满足。
前提：任何进程无权停止其他进程的运行；进程之间相对运行素的无硬性规定。

4. 什么是进程同步

进程同步是指系统中多个进程中发生的事件存在某种时序关系，需要相互合作共同完成一项任务。

5. 自旋锁（忙等锁）的优势和劣势

优点：尽量减少线程的阻塞，避免两次上下文切换的开销；
缺点：由于进程长时间占有 CPU 却不进行任何工作，导致获取锁的时间过长，自旋消耗远大于上下文切换的消耗，导致 CPU 资源的浪费。

6. Hoare 管程和 Mesa 管程的区别：

①Mesa 管程出错较少；
②Hoare 管程用 if 语句判断，Mesa 管程用 while 语句判断；
③当出现进程 P 唤醒进程 Q 的情形时，Hoare 管程会让进程 Q 执行，Mesa 管程的执行顺序是任意的。

7. 锁和条件变量

锁是一个互斥量，有两种状态，可以加锁/解锁；条件变量是条件等待。区别在于条件变量的等待用的是 if，锁的等待用的是 while。

8. 进程的基本通信机制

消息传递，共享内存，管道，套接字，远程过程调用。

操作系统期末考前整理

前言

本资料只是笔者对操作系统课程内容的部分重点整理以及部分考点的猜测，笔者不保证涵盖全部考试范围，不为本资料与考试内容的任何偏差负责。请读者基于课程 PPT 进行复习总结。

六、死锁

1. 产生死锁的必要条件

①互斥使用（资源独占）：一个资源每次只能给一个进程使用；

②占有且等待（请求和保持，部分分配）：一个进程在申请新的资源的同时保持对原有资源的占有。

③不可抢占（不可剥夺）：资源申请者不能强行从资源占有者手中夺取资源，资源只能由占有者自愿释放。

④循环等待：存在一个进程循环等待队列$\{P_1, P_2, \cdots, P_n\}$，其中 P_1 等待 P_2 占有的资源，P_2 等待 P_3 占有的资源，……P_n 等待 P_1 占有的资源，形成一个进程等待环路。

2. 资源分配图及其化简，死锁定理

资源分配图是一个有向图 $G=(V,E)$，其中 V 是结点集合，分为 P（进程）和 R（资源）；E 是有向边的集合。边(P_i, R_j)表示 P_i 请求 R_j，边(R_j, P_i)表示 P_i 拥有 R_j。

化简资源分配图时，找一个出度为 0 的进程结点，删去它关联的所有边，使之成为孤立结点，并将相应资源分配给某一等待进程（资源的入边变为出边），重复以上步骤。

死锁定理：若资源分配图中没有环路，则系统中没有死锁，若资源分配图中存在环路，则系统中可能存在死锁。若每个资源类中只包含一个资源实例（资源结点出度为 1），则环路是死锁的充分必要条件。

3. 死锁预防

①破坏“互斥使用”的条件，将独占资源变为共享资源；

②破坏“占有且等待”的条件，实现方案一为要求每个进程运行前一次性申请它所要求的所有资源，且仅当该进程所要资源均可满足时才给予一次性分配（可能导致饥饿）；实现方案二为规定一个进程在申请新的资源不能立即得到满足而变为等待状态之前，必须释放已占有的全部资源，若需要再重新申请；

③破坏“不可抢占”的条件，实现方案为虚拟化资源，允许操作系统抢占资源；

④破坏“循环等待”的条件，将资源编号，进程申请资源时必须严格按照资源编号递增次序进行。

4. 银行家算法

具体实现：进程总数 n，资源总数 m，其余变量分别为 Available[m]，Max[n][m]，Allocation[n][m]，Need[n][m]，Request[n][m]。

当进程 i 提出资源申请时，

1 若 Request[i]≤Need[i]，转 2，否则错误返回；

2 若 Request[i]≤Available，转 3，否则进程等待；

3 若分配了资源，则

Available = Available – Request[i];

Allocation[i] = Allocation[i] + Request[i];

Need[i] = Need[i] – Request[i];

安全性检查：若新分配后系统安全则分配，否则不分配，进程等待，变量为 Work[m]，Finish[n]。

1　Work = Available；Finish = **False**；
2　寻找 *i* 满足
　　Finish[*i*] == **True** && Need[*i*] ≤ Work；
　　若不存在转 4；
3　Work = Work + Allocation[*i*]；Finish[*i*] = True；转 2；
4　若所有的 Finish 均为 **True**，系统安全，否则不安全。

5. 死锁检测的时机

①进程由于资源请求不满足而等待时检测（缺点：系统开销大）；
②定时检测；
③系统资源利用率下降时检测死锁。

6. 运用死锁预防解决哲学家就餐问题

①将筷子变为共享资源，在每两位哲学家之间放两支而不是一支筷子；
②要求哲学家仅在左右筷子均空闲时才能拿起筷子，当拿起一边的筷子发现另一边无法拿起时放下筷子；
③允许抢占筷子；
④给所有哲学家编号，奇数号的哲学家必须首先拿左边的筷子，偶数号的哲学家则反之。

七、存储管理概述

1. 地址重定位

地址重定位指将用户程序中的逻辑地址转换为运行时可由机器直接寻址的物理地址的过程，其目的为保证 CPU 执行时可正确访问内存单元。
①静态地址重定位：用户程序加载到内存时一次性实现逻辑地址到物理地址的转换，一般可由软件完成；
②动态地址重定位：在进程执行过程中进行地址变换，需要 MMU 支持。

2. 地址保护的目标

①确保每个进程有独立的地址空间；
②确定进程可以合法访问的地址范围，以确保进程只访问其合法地址。

3. 空闲物理内存管理的数据结构

①位图：用 0/1 表示空闲/占用；
②空闲区表、已分配区表：表中每一项记录了空闲区/已分配区的起始地址、长度、标志；
③空闲块链表；
④伙伴系统：将内存按 2 的幂划分组织成若干空闲块链表；

4. 内存分配与回收算法

①分配算法：首次适配、下次适配、最佳适配、最差适配；
②回收算法：上相邻、下相邻、上下相邻、上下都不相邻。

5. 内存管理基本方案

单一连续区	每次只运行一个用户程序，用户程序独占内存，它总是被加载到同一个内存地址上
固定分区	把可分配的内存空间分割成若干个连续区域，每一区域称为分区。每个分区的大小可以相同也可以不同，分区大小固定不变，每个分区装一个且只能装一个进程
可变分区	根据进程的需求，把可分配的内存空间分割出一个分区，分配给该进程
页式	把用户程序地址空间划分成大小相等的部分，称为页。内存空间按页的大小划分为大小相等的区域，称为内存块（物理页面，页框，页帧）。以页为单位进行分配，逻辑上相邻的页，物理上不一定相邻
段式	用户程序地址空间按进程自身的逻辑关系划分为若干段，内存空间被动态的划分为若干个长度不相同的区域（可变分区）。以段为单位分配内存，每一段在内存中占据连续空间，各段之间可以不连续存放
段页式	用户程序地址空间：段式；内存空间：页式；分配单位：页

6. 覆盖技术

①解决的问题：程序大小超过物理内存总和；

②程序员声明覆盖结果，操作系统完成覆盖；

③不足：增加编程困难和执行时间。

7. 交换技术

①设计思想：内存空间紧张时系统将内存中某些进程暂时移到外存，把外存中某些进程换入内存，占据前者占据的区域；

②交换区：系统指定一块特殊磁盘区域作为交换空间，包含连续磁道，操作系统可以使用底层磁盘读写对其高效访问。

八、虚拟存储管理技术

1. 虚拟页式内存管理的设计思想

①装载程序时，不是装入全部页面，而是装入几个甚至零个页面；

②如果进程执行时所需页面不在内存，则动态装入所需页面；

③需要时，将内存中暂时不用的页面交换到磁盘，以便获得更大的内存空间。

2. 页表表项内容

①页框号（内存块号，物理页面号，页帧号）；

②有效位（驻留位，中断位）：该页是在磁盘还是在内存；

③访问位（引用位）；

④修改位（dirty 位）；

⑤保护位：可读/写/执行。

3. 反转页表的设计思想

从物理地址空间出发建立页表，页表项记录进程 *i* 的某虚拟地址（虚页号）与页框号的对应关系。

4. 地址转换

```
if（虚拟页面不在内存、页面非法、或者
被保护）  {
        硬件产生异常，陷入操作系统，
执行页面错误服务程序（Page Fault）
} else {
   页框号 = 页表[虚页号]
   物理地址 = 页框号  拼接  页内偏移
}
```

5. 缺页异常处理

①地址映射过程中，硬件检查页表发现访问的页不在内存，则产生缺页异常；

②操作系统执行缺页异常处理程序，获得磁盘地址，启动磁盘将该页调入内存；

③若内存中有空闲页框，则分配一页，将新调入页装入内存，修改页表中相应页表项的驻留位和页框号；

④若内存中没有空闲页框，置换某一页，若修改过则写回磁盘。

6. 页面置换中为什么要锁定某些页面

①采用虚拟内存后，程序执行时间不确定；

②通过锁定某些页面，可以不让操作系统将正在使用的页面换入内存，避免由于交换产生的不确定性延迟；

③常见的需要锁定的页面包括：操作系统核心代码、核心数据结构、I/O 缓冲区。

7. 页面置换的范围

①局部置换策略：仅在本次产生缺页的进程驻留集中选择，固定分配只能采用局部置换策略；
②全局置换策略：将内存中所有未锁定页面都作为置换的候选，可变分配既可以采用局部置换策略也可以采用全局置换策略。

8. 页面置换算法

算法	评价
OPT	不可实现，但可作为基准
NRU	LRU的很粗略的近似
FIFO	可能淘汰重要的页面
Second Chance	比FIFO有很大的改善
Clock	现实的
LRU	很优秀，但很难实现
NFU	LRU的相对粗略的近似
Aging	非常近似LRU的有效算法
Working set	实现起来开销很大
WSClock	好的有效的算法

9. 页面清除策略

①思想：保存一定数目的页框供给比使用所有内存并在需要时搜索有更好的性能；
②实现：使用双指针时钟，前指针由分页守护进程控制，指向脏页时写回磁盘，后指针与时钟算法相同，用于页面置换。此时，后指针命中干净页面的概率增加。

10. 页缓冲技术的思路

①不丢弃置换的页面，而是将它们放入两个链表中，未修改放入空闲页链表中，已修改放入修改页链表中；
②被修改的页以簇的形式写回磁盘，减少磁盘访问时间；
③被置换的页依然留在内存中，进程需要访问时再次加入驻留集（代价小）。

11. 内存映射文件

①基本思想：进程通过系统调用将一个文件映射到虚拟地址空间的一个区域，访问文件就像访问一个大数组，而不是对文件进行读写。进程退出或解除文件映射时修改页写回；
②共享库文件中的共享对象：多个进程调用共享库文件中的代码，但共享库文件在内存和磁盘只需要一个副本；
③私有的写时拷贝对象：内核将多个进程的页框状态均标记为只读，若某个进程试图写则引发 page fault，内核分配新页框并将内容拷贝，其状态修改为读写。

九、Windows 虚存管理

1. 内存管理器的组成部分

①一组执行体系统服务程序，包括工作集管理器、进程/栈交换器、修改页面写出器、映射页面写出器、零页线程等；
②一个页面错误陷阱处理程序。

2. 地址转换机制

①Windows 设置用户页表和系统页表；
②PDE、PTE 最后一位为 1 表示有效，当访问的虚拟页在内存时，PDE、PTE 均有效，CPU 根据 PDE、PTE 将虚拟地址转换为物理地址，这一过程不需要操作系统介入；
③若访问的虚拟也不在内存，此时 PTE 无效，引发 page fault。

3. 缺页异常处理

①CPU 地址转换时发现 PTE 无效，引发 page fault，产生缺页中断；
②CPU 将地址转换过程中的虚拟地址装入寄存器 CR2；
③CPU 根据中断号在中断描述符表中找到中断描述符，根据中断描述符的地址转到中断处

理程序；
④异常处理程序分析异常产生原因并相应处理。

4. 用户空间内存分配方式

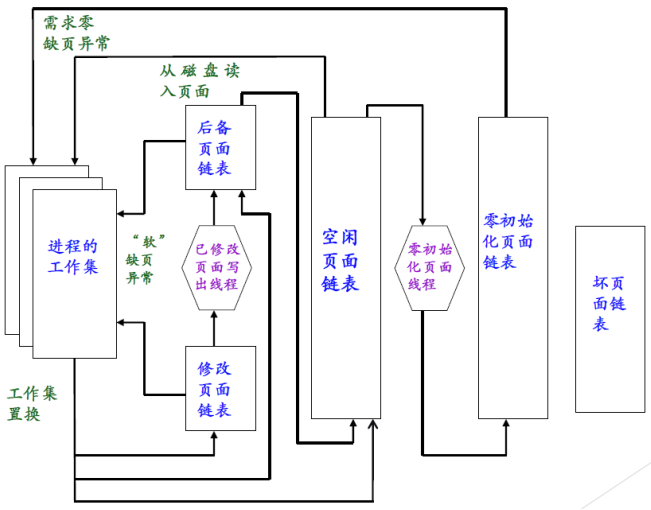
①以页为单位的虚拟内存分配：用户必须通过“保留”和“提交”两个阶段使用一定的地址范围，对于每一进程，虚拟地址描述符 VAD 描述一段被分配的进程地址空间的状态，虚拟地址描述信息构成一棵二叉树；
②利用区域对象实现内存共享文件：使用区域对象将文件映射到进程的地址空间，访问文件就像访问一个大数组访问无效页面引起缺页中断时自动将页面调入内存，若应用程序修改了页面则在常规调度时写回；
③内存堆方法：适用于大量小型内存申请。

5. 工作集

①工作集：驻留在物理内存中的虚拟页面的子集，包括进程工作集和系统工作集；
②内存管理器通过请求式页面调度算法以及簇方式将页面装入内存；
③内存访问的局部性大致稳定时，工作集大小也大致稳定，局部性区域改变时，工作集快速扩张收缩到下一个稳定值；
④页面置换算法：基于工作集模型，当可用页框降低到一定程度时启动工作集修整策略；平衡集管理器线程调用工作集管理器，周期性检查内存状态。

6. 页框的状态

①活动/有效：该页框在某个进程的工作集中，此进程对应的 PTE 有效；
②过渡：系统正在从一个文件将内容读入页框，或正在向一个文件写入该页框内容；
③空闲：该页框中的内容不再需要；
④零初始化：该页框空闲并已经被零初始化；
⑤坏：该页框硬件错误，不能使用；
⑥后备：该页框曾经在某个进程的工作集中，且该页框的内容在被此进程使用时没有改变，该页框现在已被移出该进程的工作集，但页框中的内容仍是此进程的内容，当此进程需要再次访问这一页内容时，只需要重新设定该 PTE 的标志，并把该 PTE 变为有效，即把该页框从 standby 状态变为 active(valid)状态即可；
⑦修改：该页框曾经在某个进程的工作集中，且该页框的内容在被此进程使用时有改变，该页框现在已被移出该进程的工作集，但页框中的内容仍是此进程的内容，当此进程需要再次访问这一页内容时，只需要重新设定该 PTE 的标志，并把该 PTE 变为有效，即把该页框从 modified 状态变为 active(valid)状态即可，在该页框被系统作为其他用途使用之前，需要将该页框中的内容写入硬盘中的页文件中。



7. 可执行文件的装载

①读取文件中的第一页，包含了 DOS 头、PE 文件头和段表；
②检查进程地址空间中目标地址是否可用，如果不可用，则另外选一个装载地址；
③使用段表中提供的信息，将 PE 文件中所有的段一一映射到地址空间中相应的位置；
④如果装载地址不是目标地址，则进行 Rebasing；
⑤装载所有 PE 文件所需要的 DLL 文件；
⑥对 PE 文件中的所有导入符号进行解析；
⑦根据 PE 头中指定的参数，建立初始化栈和堆；
⑧建立主线程并且启动进程。

十、文件系统

1. 文件控制块（FCB）的常用属性

FCB 是操作系统为管理文件而设置的数据结构，存放为了管理文件所需的有关信息（文件属性或元数据），常用属性包括文件名、文件号、保护、口令、创建者、当前拥有者、文件地址、大小、类型、共享计数、创建时间、最后修改时间、最后访问时间、各种标志等。

2. 实现文件系统需要考虑的内容

磁盘上：
①如何启动所存储的操作系统；
②磁盘是怎样管理的，即怎样获取磁盘的有关信息；
③目录文件在磁盘上怎样存放，普通文件在磁盘上怎样存放；
内存中：
④进程使用文件时，操作系统如何支持与管理。

3. 磁盘上的主要内容

①引导区：包括从该段引导操作系统所需要的信息，每个卷（分区）一个，通常为第一扇区；
②卷（分区）信息：包括该卷（分区）的块（簇）数、块（簇）大小、空闲块（簇）数量及指针、空闲 FCB 数量及指针；
③目录结构（目录文件）；
④用户文件。

4. 文件的物理结构、每种结构的优缺点

①连续结构（顺序）：优点为简单高效，支持顺序/随机存储，所需磁盘寻道次数和时间最少，可以同时读入多个块，检索容易，缺点为文件不能动态增长，不利于文件内容插入删除，会出现外部碎片问题；
②链接结构（指针连接）：优点为提高了磁盘空间利用率，不存在外部碎片问题，有利于文件内容动态插入/删除/扩充，缺点为存取速度慢，不适用于随机存取，存在可靠性问题（指针出错），有更多的寻道次数和时间，指针会占用空间；
③索引结构（系统为每个文件建立索引表，必要时可有二/三级索引）：优点是保持了链接结构的优点，能顺序/随机存取，满足了文件动态增长/插入/删除的要求，能充分利用磁盘空间，缺点为较多寻道次数和时间，索引表本身存在开销。

5. UNIX 文件系统

UNIX 文件系统采用多级索引结构，每个文件的索引表有 15 个索引项，每项 2 字节，其中前 12 个索引项用于直接索引，第 13~15 个索引项分别用于一级、二级、三级索引。若每个块可以存放 x（一般为 256）个索引项，则 UNIX 多级索引结构中，文件最大可以达到 (x^3+x^2+x+12)个物理块。

UNIX 文件系统的 FCB 包括目录项和 i 结点，i 结点记录了每个文件索引表所在的块。

对于目录文件，索引表分别记录了当前目录、上一级目录和目录下一级所有文件/目录的 i 结点。

引导记录	超级数据块	空闲区管理	i 结点区	根目录区	文件和目录区

6. Windows-FAT16 文件系统

FAT16 使用文件分配表（FAT）记录簇的分配状态。文件分配表可以看成一个大的整数数组，每个元素都是整数，记录该文件下一簇的簇号（若有）或为-1（若为当前文件最后一块）。簇号从 0 开始编号，但 0 和 1 一般保留。目录文件、普通文件的记录方式与 UNIX 类似，但不采用多级索引结构。

引导区	文件分配表 1	文件分配表 2	根目录文件	其他目录文件和普通文件

7. 成组链接法分配与回收算法

①分配：查 L 单元（空闲块数）内容，若空闲块数大于 1 则 i 赋值为 $L+1$，从 i 单元得到一个空闲块分配，并将空闲块数减 1；若空闲块数为 1 则取出 $L+1$ 单元内容，将该块内容复制到专用块，将专用块内容读到内存 L 开始的区域，并将该块分配；若空闲块数为 0 则进程等待；

②回收：查 L 单元（空闲块数）内容，若小于 100 则空闲块数加 1，将 j 赋值为空闲块数 + L 并将归还块号填入 j 单元；若等于 100 则把内存中登记的信息写入归还块，将归还块号填入 $L+1$ 单元，并将 L 置为 1。

8. create 创建文件的过程

①创建文件的实质是建立 FCB。操作系统检查参数的合法性，在目录中为新文件建立一个目录项，根据参数提供有关内容；

②为文件申请必要的磁盘块。

9. open 打开文件的过程

①根据文件路径查询目录，得到目录项（i 结点号）；

②根据文件号查系统打开文件表看文件是否已经打开，若是则共享计数加 1，若否则将目录项、i 结点信息等填入系统打开文件表空表项，将其共享计数设为 1；

③根据打开方式、共享说明和用户身份检查访问合法性；

④在用户打开文件表中取一空表项，填写打开方式等，并指向系统打开文件表对应表项。

10. read 读文件的过程

①根据打开文件时得到的文件描述符，找到相应的文件控制块（目录项），确定读操作的合法性：读操作合法则进入②，否则出错处理；

②将文件的逻辑块号转换为物理块号，根据参数中的读指针、长度与文件控制块中的信息，确定块号、块数、块内位移；

③申请缓冲区；

④启动磁盘I/O操作，把磁盘块中的信息读入缓冲区，再传送到指定的内存区（多次读盘）；

⑤反复执行③、④直至读出所需数量的数据或读至文件尾。

11. 文件共享的两种方式

①硬链接：利用多个路径名描述同一共享文件，多个目录项指向一个文件；

②软链接：建立特殊类型的文件，内容是要共享的文件的路径名，只有真正的拥有者才有 i 结点指针。优点为计算机网络环境下可用，缺点为系统开销大，文件路径可能成环。

12. 文件系统的管理及手段

①可靠性：抵御和预防各种物理和人为破坏的能力，解决方法为备份（全量/增量转储，物理/逻辑转储）；

②一致性：确保磁盘块和目录系统内容一致，解决方法为设计程序，系统再次启动时运行，检查磁盘块和目录系统是否一致；
③安全性：确保未经授权的用户不能存取某些文件，解决方案为文件保护，包括用户身份验证和访问控制。

13. UNIX 的文件保护

UNIX 采用文件的二级存取控制，审查用户权限和操作合法性。
①对访问者的识别：将用户分类为 owner/group/other；
②对操作权限的识别：将操作分类为 r/w/x/-。

14. Windows 的文件访问方式

①用户可以使用文件缓冲，通过异步的方式使处理器和 I/O 并发工作；
②用户对磁盘的访问通过文件缓存实现，系统 cache manager 实现对缓存的控制，读取时预取，LRU 更新，定期写回（1 秒）；
③写回机制：用户对磁盘写数据时只更改 cache，由 cache manager 决定何时写回（lazy writer）；

15. 磁盘调度算法

①先来先服务：优点为简单公平，缺点为效率不高；
②最短寻道时间优先：优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先，优点为改善了磁盘平均服务时间，缺点为造成某些请求长期得不到服务；
③扫描算法（SCAN）：当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复；
④单向扫描算法（C-SCAN）：总是从 0 号柱面开始向里扫描，到达最后一个柱面后带动读写磁头快速回到 0 号柱面，返回时不为任何等待者服务，返回后可再次扫描；
⑤N-step-SCAN：将请求划分为若干长为 N 的队列，对每个队列使用 SCAN 算法；
⑥FSCAN：使用两个子队列，扫描开始时所有请求都在一个子队列中，扫描过程中所有新来请求加入另一队列；
⑦旋转调度算法：根据延迟时间决定调度。对于同一/不同磁头上的不同扇区，让首先到达读写磁头的扇区进行操作，对于不同磁头上的相同扇区，从中任选一个操作。

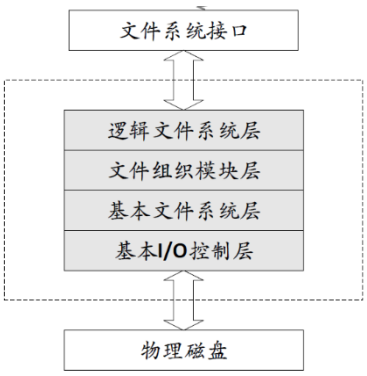
16. RAID 技术

①RAID0：条带化，无冗余，性能最佳；
②RAID1：镜像，最大程度保证数据安全和可恢复性；
③RAID2：并行访问——海明码校验，数据条带化分布于不同磁盘，在磁盘阵列中间隔写入海明码；
④RAID3：交错位奇偶检验，以字节为单位进行数据划分，设置存储校验盘保存校验码；
⑤RAID4：交错块奇偶检验，以块为单位；
⑥RAID5：交错块分布式奇偶检验，奇偶校验分散在各个磁盘，数据读出效果好但写入效果一般，磁盘利用率较好但有写损失；
⑦RAID6：交错块双重分布式奇偶检验，在 RAID5 的基础上设置两个校验码，增强了数据恢复能力，但降低了磁盘利用率和写能力；
⑧RAID7：最优化异步高 I/O 速率及高传输速率，每个磁盘都有独立 I/O 通道

17. 文件系统的设计要求

①如何定义文件系统与用户的接口（文件及属性、文件操作、目录结构）；
②如何将逻辑文件系统映射到物理磁盘设备（数据结构与算法）；
③文件系统实现时如何分层。

18. 文件系统通用模型、各层作用



①文件系统接口：定义了一组使用和操作文件的方法；
②逻辑文件系统层：使用目录结构为文件组织模块提供所需信息，并负责文件保护和安全；
③文件组织模块层：负责对具体文件以及这些文件的逻辑块和物理块进行操作；
④基本文件系统层：主要向相应的设备驱动程序发出读写磁盘物理块的一般命令；
⑤基本 I/O 控制层：由设备驱动程序和中断处理程序组成，实现内存与磁盘系统间的信息传输。

十一、IO 管理

1. I/O 指令的两种形式以及它们的区别

①I/O 独立编址中，分配给系统中所有端口的地址空间完全独立，与内存地址空间无关，使用专门的 I/O 指令对端口进行操作。其优点在于外设不占用内存的地址空间，编程时易于区分是对内存操作还是对 I/O 操作，缺点为 I/O 端口操作的指令类型少，操作不灵活；
②内存映射编址中，分配给系统中所有端口的地址空间与内存的地址空间统一编址，将 I/O 端口看作一个存储单元，对 I/O 的读写操作等同于对内存的操作。其优点在于凡是可对内存操作的指令都可对 I/O 端口操作，不需要专门的 I/O 指令，I/O 端口可占有较大地址空间，缺点为占用内存空间。
③除此以外，内存映射编址不需要特殊的保护机制来阻止用户进程执行 I/O 操作，但对一个设备控制寄存器不能进行高速缓存，且只存在一个地址空间时，所有的内存模块和所有 I/O 设备都必须检查所有内存引用，以了解由谁做出响应。

2. 以下的代码会出现什么问题，如何解决

```
LOOP: TEST PORT-4    //检测端口4是否为0
   BEQ READY         //如果为0，转向READY
   BRANCH LOOP       //否则，继续测试
READY:
```

①由于以上代码通过 TEST 指令测试一个控制寄存器是否为 0，所以它是内存映射 I/O，将对 I/O 的读写操作等效为对内存的操作。
②但是，当高速缓存有效时，第一次引用 PORT-4 导致它被高速缓存，所以之后的引用只会从高速缓存中读值，不会再查询设备。因此，只要第一次 PORT-4 端口不为 0，之后设备就绪时软件就无法发现这一点。
③解决方案为硬件必须针对每个页面具备选择性禁用高速缓存的能力。对于以上的代码，设备控制寄存器不能进行高速缓存。

3. 以某 I/O 设备为例说明 I/O 软件的设计思想

①I/O 软件的设计思想为分层，将 I/O 软件组织成多个层次，较低层考虑硬件特性，并向较高层软件提供接口，较高层不依赖于硬件，并向用户提供一个友好的、清晰的、简单的、功能更强的接口。

②用户进程层执行输入输出系统调用，对I/O数据进行格式化，为假脱机输入/输出作准备；
③独立于设备的软件实现设备的命名、设备的保护、成块处理、缓冲技术和设备分配；
④设备驱动程序设置设备寄存器、检查设备的执行状态；
⑤中断处理程序负责I/O完成时，唤醒设备驱动程序进程，进行中断处理；
⑥硬件层实现物理 I/O 的操作。

4. 引入缓冲技术解决了什么问题/为什么要引入缓冲技术

①解决CPU与I/O设备之间速度的不匹配问题；
②提高CPU与I/O设备之间的并行性；
③减少了I/O设备对CPU的中断请求次数，放宽CPU对中断响应时间的要求。

5. I/O 进程的工作流程

①I/O 进程专门处理系统中的 I/O 请求和 I/O 中断工作，对于 I/O 请求，用户程序将 I/O 请求发送给 I/O 进程后将自己阻塞，系统用 wakeup 唤醒 I/O 进程，对于 I/O 中断，内核的中断处理程序发消息给 I/O 进程，由 I/O 进程判断并处理中断；
②I/O 是系统进程，往往优先级最高，唤醒后首先关闭中断，然后接收消息；
③若没有消息，则打开中断，将自己阻塞；
④若有消息，判断消息类型。对于 I/O 请求，I/O 进程准备通道程序并发出启动 I/O 指令，对于 I/O 中断，若是正常中断则唤醒要求进行 I/O 操作的进程，若是异常中断则转入错误处理程序。

6. 提高 I/O 性能的方法

①通过缓冲技术，减少或缓解速度差距；
②通过异步 I/O，让 CPU 不等待 I/O；
③通过 DMA、通道，让 CPU 摆脱 I/O 操作。

7. 异步 I/O 的基本思想

系统实现：
①通过切换到其他线程保证CPU利用率；
②对少量数据的I/O操作会引入切换的开销；
用户实现：
③将访问控制分成两段进行；
④发出读取指令后继续做其他操作；
⑤当需要用读入的数据的时候，再使用wait命令等待其完成；
⑥不引入线程切换，减少开销。